Санкт-Петербургский государственный университет

Математико-механический факультет
Кафедра Информационно-аналитических систем
Математическое обеспечение и администрирование информационных систем

Бусаров Вячеслав Геннадьевич

Сравнительный анализ алгоритмов поиска частых наборов и их использование

Бакалаврская работа

Научный руководитель:
канд. ф.-м. н., доцент Графеева Н. Г.

Рецензент:
науч. сотр. СПИИРАН Самойлов В. В.

Санкт-Петербург
2017

SAINT-PETERSBURG STATE UNIVERSITY

Mathematics and Mechanics Faculty
Department of Analytical Information Systems
Software and Information Systems Administration

Vyacheslav Busarov

Comparative analysis of algorithms for mining frequent itemsets and their usage

Graduation Thesis

Scientific supervisor:
associate professor Natalia Grafeeva

Reviewer:
researcher SPIIRAS Vladimir Samoylov

Saint-Petersburg
2017

Оглавление

Введение

В настоящее время Big Data становятся неотъемлемой частью личной и профессиональной жизни людей. Сегодня по-настоящему успешный бизнес уже немыслим без анализа данных, а в медицине различные инструменты Data Mining используются повсеместно.

Данная работа стартовала с практической задачи профилирования крупной московской сети ресторанов: необходимо было на основе информации о деятельности каждого из ресторанов сети выделить некую специализацию каждого заведения – наборы наиболее востребованных там блюд. Ранее решение данной задачи не было автоматизировано, и её приходилось кропотливо решать вручную, что делало точный ответ невозможным[2].

Задача, наиболее близкая к данной, – это Market Basket Analysis, автором которой стал Rakesh Agrawal в 1993 году[1]. Она подразумевает выделение нестрогих правил, по которым клиенты, приобретающие некий набор товаров, зачастую покупают что-то ещё в дополнение к имеющимся покупкам. Правила такого вида именуются ассоциативными, а вероятность, с которой выбор одних товаров влечёт выбор других, называется значимостью (confidence). Задача анализа продуктовой корзины являлась предпосылкой к появлению области ассоциативных правил, однако в дальнейшем этот инструмент начали использовать для увеличения перекрёстных продаж (cross-sell)[5] и продаж с повышением цены (up-sell)[5], а также для более эффективной прямой адресной рассылки рекламных предложений (direct mail)[10].

Основная часть любого алгоритма поиска ассоциативных правил состоит в поиске часто встречающихся наборов. При условии наличия решения этой подзадачи, получение ассоциативных правил осуществляется тривиальным образом, в то время как сам поиск частых наборов вычислительно сложен. В действительности, проблема, к которой сводится поиск ассоциативных правил, является гораздо более востребованной и применяется ещё и для задач классификации и прогнозирования, например, при диагностике заболеваний сердца[8].

Исходя из значительного количества исследований в области часто встречающихся наборов и наличия множества свежих публикаций[3][2], одна из которых датируется февралём 2017 года[6], можно с уверенностью сделать вывод об актуальности проблем, сводящихся к поиску частых наборов.

Задача существует уже более 20-ти лет, и за это время появилось огромное количество алгоритмов, решающих её с той или иной степенью эффективности[4]. Однако сложность состоит в том, что до появления данной работы не существовало полноценного сравнительного анализа одновременно всех существующих алгоритмов поиска частых наборов в равных условиях[4]. При решении конкретной задачи важно иметь возможность осознанного выбора алгоритма с наибольшей производительностью, однако это невозможно было сделать при отсутствии полномасштабного исследования.

Постановка задачи

Цель данной работы — произвести сравнительный анализ алгоритмов поиска частых наборов, выбрать наиболее оптимальный из них для решения задачи профилирования крупной сети ресторанов. Для достижения этой цели в рамках данной работы были сформулированы следующие задачи:

- создать методологию сравнительного анализа;
- провести сравнительный анализ алгоритмов поиска частых наборов;
- провести анализ результатов;
- применить результаты исследования при профилировании сети ресторанов.

1. Обзор

1.1. Терминология

Как уже было отмечено, первоначальной мотивацией к созданию данной работы стала задача профилирования крупной сети ресторанов. Давайте формализуем задачу профилирования, обобщив её до общего случая. Пусть имеется множество объектов с общим для всех словарём упорядоченных признаков $I = \{i_1, i_2, ..., i_m\}$, а также некоторое количество наборов признаков из словаря $T^n = \{\tau_1, \tau_2, ..., \tau_n | \tau \subset I\}$, каждый из которых описывает тот или иной объект. Тогда профилем назовём такое подмножество словаря $P \subset I$, которое наиболее ёмко характеризует данный объект, то есть достаточно часто встречается в его исходных описаниях T^n. Порог частоты встречаемости будет определяться входными данными, по умолчанию будет взято наиболее часто встречающееся подмножество словаря.

Поскольку задача профилирования объектов не была автоматизирована ранее, мной была рассмотрена наиболее близкая к ней задача Market Basket Analysis, сформулированная известным учёным Rakesh Agrawal в 1993 году[1]. В этой же работе была определена область ассоциативных правил.

- Имея упорядоченный набор признаков $I = \{i_1, i_2, ..., i_m\}$, а также набор исходных транзакций (описаний) $T^n = \{\tau_1, \tau_2, ..., \tau_n | \tau \subset I\}$, для каждого набора признаков $s \subset I$ положим $s(\tau) = 1$, если признаки из s совместно встречаются в $\tau \in T^n$.

- Поддержкой (support) набора s назовём $v(s) = \frac{1}{n} \sum_{i=1}^{n} s(\tau_i)$.

- Набор $s \subset I$ назовём частым (frequent itemset), если его поддержка превышает минимальную поддержку — входной параметр δ (MinSupp), то есть $v(s) \geq \delta$.

- Ассоциативное правило (association rule) $x \to y$ — это пара непересекающихся наборов $x, y \subset I$ таких, что

1. наборы x и y совместно часто встречаются, $v(x \cup y) \geq \delta$,
2. если в описании встречается x, то с вероятностью, превышающей σ, встречается также и y, $v(y|x) = \frac{v(x \cup y)}{v(x)} \geq \sigma$,
   при этом $v(y|x)$ — значимость правила (confidence), σ — минимальная значимость (MinConf), входной параметр.

Главной и наиболее вычислительно сложной частью задачи поиска ассоциативных правил является поиск часто встречающихся наборов. Найдя их, из каждого полученного частого набора путём разбиения его на два непересекающихся поднабора можно за линейное от количества признаков время выделить ассоциативное правило, отбросив те из них, чья значимость меньше заданной минимальной σ[4]. Таким образом имеет смысл сравнивать только алгоритмы поиска часто встречающихся наборов, на сегодняшний день опубликовано 12 таковых.

Существует два основных подхода к решению данной задачи: «генерация кандидатов и тестирование» (candidate-generation-and-test) и «наращивание шаблонов» (pattern-growth). Представители первого из них генерируют наборы длины $k + 1$, основываясь на наборах блины k, и отсекают некоторые ветки перебора по свойству антимонотонности: поддержка набора признаков не превосходит поддержки любого его поднабора. Алгоритмы, представляющие второй подход, осуществляют поиск рекурсивно: разбивают базу данных на части и ищут локальные ответы, которые в дальнейшем наращиваются до общего результата[4].

1.2. Apriori

[4] Данный алгоритм был предложен автором тематики ассоциативных правил Rakesh Agrawal в 1994 году и является одним из первых решений данной задачи. Он лёг в основу подхода «генерации кандидатов и тестирования».

Apriori использует поуровневый принцип обхода в ширину для перебора всех возможных комбинаций наборов разной длины:

1. $G_1 := \{\{i\} | i \in I, v(\{i\}) \geq \delta\}$ – множество частых наборов длины 1;

2. $G_j := \{s \cup \{i\} | s \in G_{j-1}, i \in G_1 \backslash s, v(s \cup \{i\}) \geq \delta\}$ – множество часто встречающихся наборов длины j, генерируемое из наборов длины $(j-1)$;

3. свойство антимонотонности используется для отсечения некоторых ветвей перебора: если при некотором j множество $G_j = \emptyset$, то дальнейшая работа с ним прекращается.

1.3. FP-Growth

[4] Данный алгоритм был опубликован в 2000 году за авторством J.Han. Решение стало родоначальником подхода «наращивания шаблонов», впервые отвергнув перебор.

FP-Growth, как и многие другие алгоритмы, сводит задачу поиска частых наборов к задаче хранения словаря, для решения которой в данном случае используется префиксное дерево – FP-Tree. Предварительно признаки внутри каждой транзакции сортируются по убыванию значения поддержки, их порядок фиксируется. Нечастые одноэлементные наборы отбрасываются заранее. Сама структура данных строится по следующим принципам:

- корень дерева v_0 содержит значение $null$;
- в каждой вершине v дерева T задаются: признак $i_v \in I$, множество дочерних вершин $C_v \subseteq T$, поддержка $p_v = v(s_v) : s_v = \{i_u | u \in [v_0, v]\}$, $[v_0, v]$ – путь от корня дерева v_0 до вершины v;
- $V(T, i) = \{v \in T | i_v = i\}$ – все вершины признака i;
- $P(T, i) = \sum\limits_{v \in V(T,i)} p_v$ – суммарная поддержка признака i;
- дерево подразделяется на уровни, каждый из которых соответствует какому-то признаку и каждому признаку сопоставляется единственный уровень. При этом следующая на пути вершина может находиться на любом уровне ниже текущего, так как и те, и

другие упорядочены по убыванию поддержки соответствующих им элементов.

В процессе работы алгоритма неоднократно формируется условное FP-дерево (CFP-Tree) – это FP-Tree, построенное только по транзакция, содержащим данный признак. Пусть дано FP-дерево T и признак $i \in I$. Conditional FP-Tree $T' = T|i$ будет получено, если:

1. оставить в дереве только вершины на путях из вершины v признака i снизу вверх до корня v_0, то есть $T' := \bigcup\limits_{v \in V(T,i)} [v, v_0]$;
2. увеличить значения поддержек p_v вершин $v \in V(T', i)$ снизу вверх по правилу $p_u := \sum\limits_{w \in C_u} p_w$ для всех $u \in T'$;
3. удалить из T' все вершины, соответствующие признаку i, так как к текущему моменту все признаки, лежащие ниже i, уже будут рассмотрены.

Заметим, что T' формируется из дерева T без обращения к базе транзакций.

Формирование итогового ответа происходит посредствам рекурсивной процедуры, принимающей FP-дерево T, набор $s \subseteq I$ и список наборов R. В результате работы все частые наборы, содержащие s, будут добавлены в R. Перебираются все признаки $i \in I | V(T, i) \neq \emptyset$ по уровням снизу вверх, и если $P(T, F) \geq \delta$, то:

1. $R := R \cup \{s \cup \{i\}\}$;
2. построить $T' = T|i$;
3. запустить данную процедуру от новых параметров: $T', s \cup \{i\}, R$.

Проиллюстрируем вышеописанное построение FP-Tree конкретным примером (рис. 1). Имеется некоторое количество транзакций, которые после обозначения признаков некоторыми символами (или наборами символов) преобразуются в слова. Предварительно подсчитывается

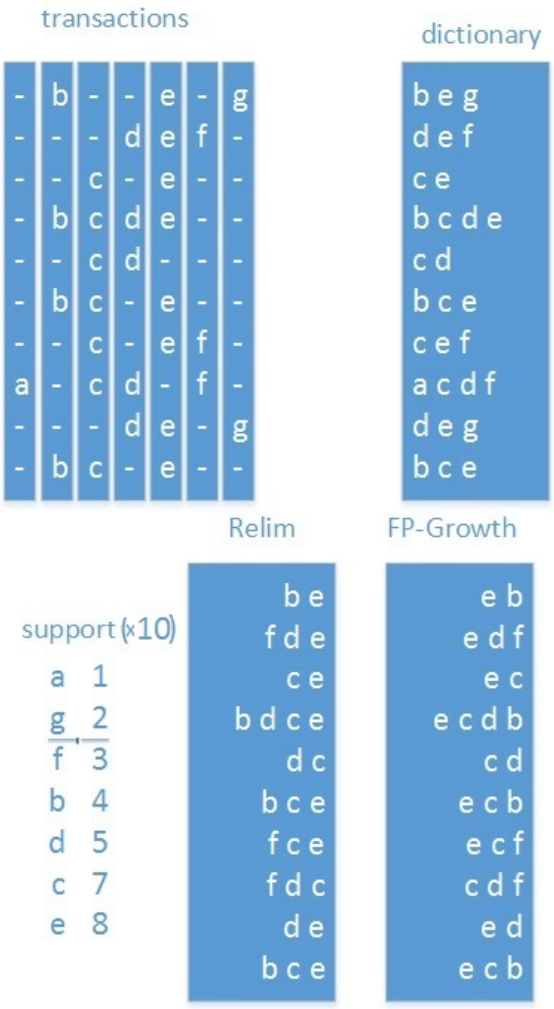

Рис. 1: Пример первого этапа работы алгоритмов FP-Growth и Relim[4]

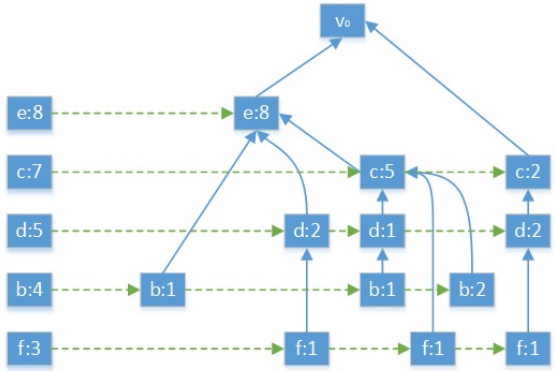

Рис. 2: Пример исходного FP-дерева[4]

поддержка каждого одноэлементного набора. Нечастые наборы отсеиваются, остальные упорядочиваются внутри каждой транзакции в порядке не убывания значения поддержки. Для простоты положим минимальную поддержку $\delta = 0.3$. За один проход по рассматриваемой базе транзакций будет построено дерево следующего вида (рис. 2).

1.4. Relim

[4]Данный алгоритм опубликован в 2005 году в работе известного аналитика Christian Borgelt. Название алгоритма происходит от его

принципа работы: «REcursive ELIMination scheme» (схема рекурсивного отсеивания). Relim осуществляет поиск всех частых наборов с данным префиксом, рекурсивно увеличивая его вместе с обновлением значения поддержки. Он является представителем подхода «наращивания шаблонов». Основные аспекты алгоритма, проиллюстрированные на конкретном примере (рис. 1), описаны далее.

1. Первая итерация по исходным данным позволяет посчитать абсолютную поддержку (частоту встречаемости) каждого признака в отдельности и исключить все те из них, которые не являются частыми. Свойство антимонотонности гарантирует, что эта процедура не повлияет на точность ответа. Для корректной работы на дальнейших этапах необходимо отсортировать признаки каждой транзакции по неубыванию их значений поддержки. В рассматриваемом примере минимальное значение поддержки $\delta = 0.3$.

2. На следующей итерации строится массив списков транзакций, заголовком каждого из которых является признак, с которого начинаются все транзакции данного списка.

3. В соответствующую ячейку массива первоначально записывается длина списка, хранящегося в ней. Однако в дальнейшем там будет находиться количество транзакций, хранящихся в списке данной ячейки массива и содержащих данный признак-заголовок.

4. Помимо массива списков на каждой итерации алгоритма хранится общий префикс всех рассматриваемых на данном этапе транзакций. При этом транзакции хранятся, начиная с первого признака, не входящего в префикс.

5. Основной принцип алгоритма: количество транзакций, содержащих набор $s \cup i_k | s \subseteq I, i_k \in I$, равно количеству транзакций, содержащих i_k в контексте префикса s. Очевидно, что поддержка набора – это отношение количества содержащих его транзакций к общему количеству транзакций.

6. Поиск частых наборов осуществляется рекурсивной процедурой от следующих параметров: (1)структура, построенная по вышеописанному принципу; (2)текущий префикс (первоначально пустой), (3)минимальная поддержка.

7. Процедура движется в двух направлениях: цикл по данной структуре данных и рекурсивный спуск – вызов процедуры из самой себя.

    - В цикле каждая ячейка массива рассматривается в отдельности, и строится новая структура данных (описанного выше формата) по списку транзакций данного признака-заголовка i_k.
    - Вычисляется поддержка набора $s \cup i_k$ – объединения текущего префикса и рассматриваемого признака-заголовка. Если набор является частым, он добавляется к ответу.
    - Рассматриваемый признак-заголовок добавляется к текущему префиксу и запускается рекурсивная процедура от полученной структуры данных и обновлённого префикса.
    - Опустошается список рассматриваемого признака-заголовка.
    - Поэлементно объединяются списки двух массивов: текущего и полученного для рекурсивного вызова.
    - Осуществляется переход цикла к следующего признаку-заголовку.

Пример работы цикла обхода текущей структуры данных проиллюстрирован на рис. 3. Здесь на каждой итерации светлым выделены перемещённые элементы списка, справа изображены заново сгенерированные массивы, от которых и будут производиться рекурсивные вызовы. На рис. 4 рассмотрено получение набора 'dc' как часто встречающегося с поддержкой 0.3 на шаге 4.1.1 и переход к шагу 4.2, сразу после которого будет получен набор 'de' с поддержкой 0.3.

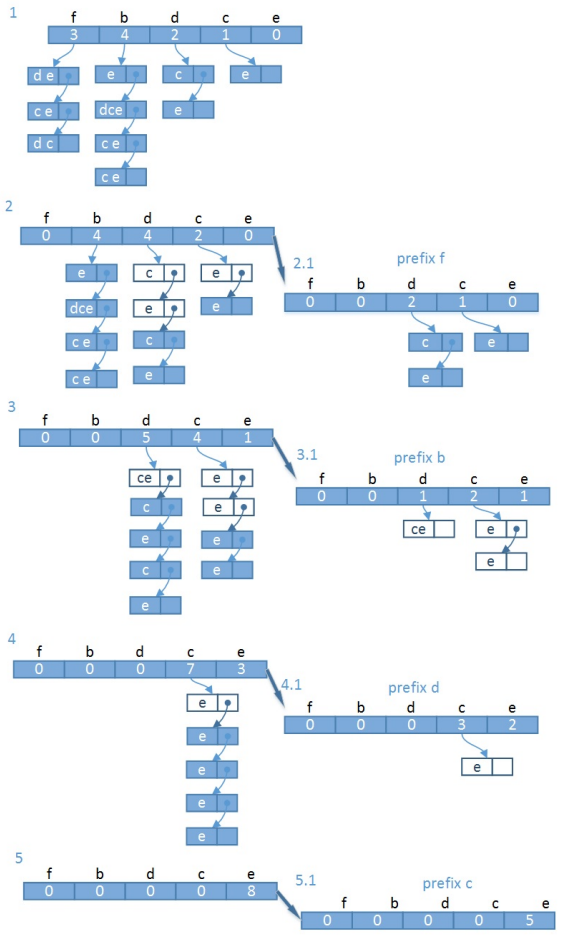

Рис. 3: Пример циклически изменяющейся структуры данных алгоритма Relim[4]

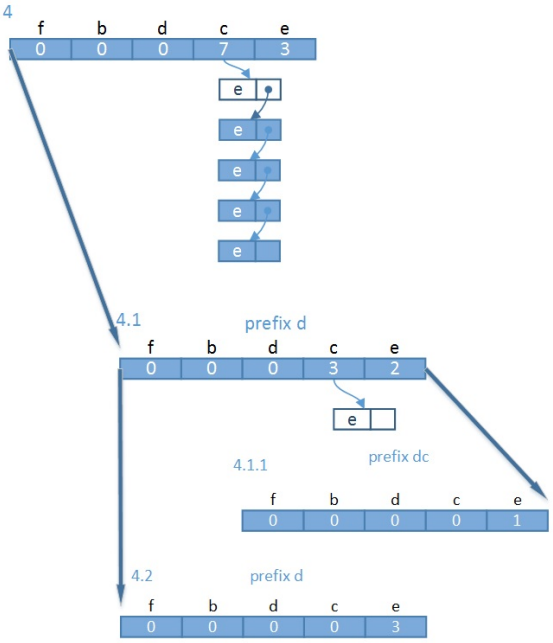

Рис. 4: Пример рекурсивного перехода алгоритма Relim[4]

1.5. PrePost+

[4]Алгоритм был предложен в 2015 году, его автором стал Z. H. Deng. Для решения задачи PrePost+ использует сразу 3 структуры данных: N-list, PPC-tree и Set-enumeration tree. Это так называемый «Apriori-like» алгоритм, то есть он является представителем подхода «генерация кандидатов и тестирование».

PPC-tree - префиксное дерево (похожее на FP-tree), каждая вершина которого содержит: (1)имя признака, (2)счётчик, хранящий количество транзакций, лежащих на пути из корня, до данной вершины, (3)список потомков, (4)pre-order и (5)post-order. Последние два значения – порядковые номера вершины при прямом и обратном обходе де-

рева. Они и дают название алгоритмам PrePost и PrePost+ и могут быть получены, например, как время входа в вершину и выхода из неё при обходе в глубину (DFS) из корня. Корень же всегда содержит только значение null. Пример PPC-дерева, построенного на рассмотренных ранее данных (рис. 1), изображён на рис. 5

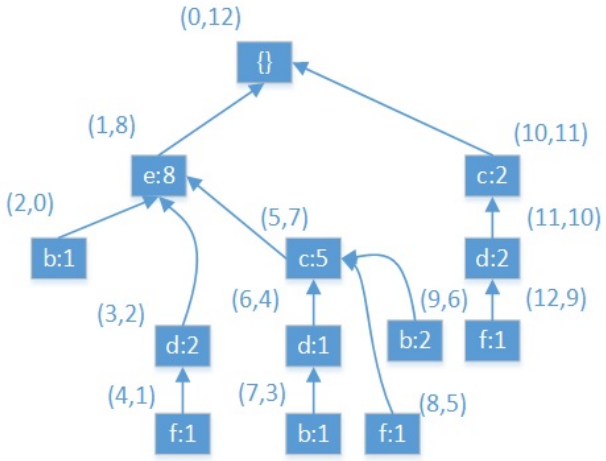

Рис. 5: Пример PPC-tree[4]

В ходе работы алгоритма для каждой вершины N префиксного дерева формируется $PP - code$, имеющий следующий вид:

$$< (N.pre - order, N.post - order) : count >$$

N-list часто встречающихся наборов представляет из себя список PP-кодов вершин PPC-tree, упорядоченных по неубыванию значений pre-order. N-list, состоящий из k и более вершин, – это пересечение N-list соответствующих признаков.

Set-enumeration tree – дерево, содержащее все возможные наборы признаков, упорядоченный по неубыванию значений поддержки. Каждая вершина хранит единственный часто встречающийся признак.

В процессе подготовки базы данных к работе алгоритма, из неё удаляются нечастые признаки. Все элементы внутри каждой транзакции упорядочиваются по невозрастанию поддержки. При этом очевидным образом находятся все частые наборы длины 1 и 2.

Алгоритм основан на двух свойствах:

1. по данному N-list набора $s \subseteq I$ из k признаков $\{< (x_1, y_1) : z_1 >, < (x_2, y_2) : z_2 >, ..., < (x_k, y_k) : z_k >\}$ можно получить значение поддержки как $v(s) = \sum_{i=1}^{k} z_i$;
2. $\forall s \subseteq I \forall i \in I : v(s) = v(s \cup i) \rightarrow \forall A \subseteq I : A \cap s = \emptyset, i \notin A$ верно $v(s \cup A \cup f) = v(s \cup A)$. Действительно, если $v(s) = v(s \cup i)$, то любая транзакция, содержащая s, содержит также и i, из чего очевидным образом следует вышеописанное тождество.

Основное отличие алгоритмов PrePost и PrePost+ заключается в способе отсечения кандидатов, претендующих на звание частых наборов. PrePost использует свойство единственности пути в N-list, в то время как именно свойство №2 используется в модифицированном алгоритме 2015 года для отсечения сгенерированных кандидатов. Для этого отдельно хранятся все эквивалентные наборы, то есть те, величины поддержки которых совпадают.

Первоначально имея множество F_2 всех частых наборов длины 2, PrePost+ для каждого $s \in F_2$ ищет все частные наборы, содержащие s. При этом каждый раз генерируя новое Set-enumeration tree и N-list, как пересечение N-листов признаков данного s. В процессе обхода этих структур данных выявляются частые наборы с заявленным свойством, а также эквивалентные множества признаков. Таким образом получается F_3, с которым проделывается всё то же самое, что и с F_2. Операция повторяется до тех пор, пока $F_j \neq \emptyset$. Результат $F = \bigcup_{F_j \neq \emptyset} F_j$.

1.6. Apriori Hybrid

год публикации: 1994

подход: «генерация кандидатов и тестирование»

используемые структуры данных:

Hash-tree или Hash-table для хранения генерируемых кандидатов (упрощает подсчёт поддержки новых наборов). И двумерные массивы

элементов числового типа для хранения часто встречающихся наборов, каждый из которых имеет свой id. По ним и происходит индексация.

некоторые ключевые моменты:

Данный алгоритм работает по тем же основным принципам, что и Apriori, однако он не посещает исходную базу транзакций для подсчёта поддержки каждого рассматриваемого набора. Вместо этого используется хеширование и пересечение имеющихся наборов[4].

1.7. Eclat

год публикации: 2000

подход: «генерация кандидатов и тестирование»

используемые структуры данных:

Lattices – частично упорядоченные множества («partially ordered sets»), в которых каждая пара элементов имеет уникальные supremum и infinum.

некоторые ключевые моменты:

Все кандидаты хранятся в этой структуре данных – Lattices, и для получения ответа алгоритм осуществляет различные обходы, основанные на поиске в ширину или в глубину. Одной из основных эвристик является разбиение наборов на подмножества для обособленного рассмотрения каждого из них. Также в процессе поиска частых наборов Eclat пытается выделить классы эквивалентности, чтобы таким образом сузить диапазон перебора[4].

1.8. dEclat

год публикации: 2003

подход: «генерация кандидатов и тестирование»

используемые структуры данных:

Diffset – структура данных, хранящая любые множества, со встроенными функциями пересечения и объединения.

некоторые ключевые моменты:

dEclat является модификацией алгоритма Eclat тех же авторов. Основным изменением стало использование новой структуры данных, позволяющей отсечь большее количество кандидатов. В данном решении вновь вводится понятие эквивалентности, однако здесь оно основано на значениях некоторой функции на множестве наборов $\{s|s \subseteq I\}$. Один diffset хранит классы эквивалентности, другой – префиксы кандидатов разной длины[4].

1.9. LCMFreq v.2/v.3

год публикации: 2005

подход: «наращивание шаблонов»

используемые структуры данных:

Комбинированное использование общеизвестных bitmap, prefix tree и array list.

некоторые ключевые моменты:

Префиксное дерево содержит возможных кандидатов на звание частых наборов, при этом порядок признаков чётко фиксируется. Исходные транзакции хранятся в array list. Bitmap используется для более эффективного подсчёта поддержки. Разные версии алгоритма варьируют совместное использование этих структур данных. V.3 была признана авторами более эффективной, её я и использовал для экспериментов, в дальнейшем именуя просто LCMFreq[4].

1.10. H-mine

год публикации: 2007

подход: «наращивание шаблонов»

используемые структуры данных:

H-struct – структура данных, хранящая часто встречающиеся признаки вместе со ссылками на содержащие их транзакции в исходной базе данных. Способ её построения схож с FP-tree, однако вместо явного

хранения транзакций алгоритм оперирует со ссылками на них. Благодаря этому H-mine превосходит многие алгоритмы по эффективности использования памяти.

некоторые ключевые моменты:

Основная идея обхода H-mine в целом совпадает с FP-Growth. Принципиальное отличие этих алгоритмах заключается в выбранном подходе к хранению исходных данных, что влечёт изменение производительности[4].

1.11. PPV

год публикации: 2010

подход: «генерация кандидатов и тестирование»

используемые структуры данных:

PPC-tree – префиксное дерево, каждая вершина которого содержит: (1)имя признака, (2)счётчик, хранящий количество транзакций, лежащих на пути из корня, до данной вершины, (3)список потомков, (4)pre-order и (5)post-order (рис. 5).

Node-list – структура данных, состоящая из PP-кодов, извлечённых из PPC-tree. Она абсолютно аналогична N-list с той лишь только разницей, что первая использует потомков для представления набора, а вторая предков.

некоторые ключевые моменты:

В PPC-tree хранятся исходные транзакции. Рассматриваемые наборы содержатся в Node-list, что позволяет с быстрее подсчитывать поддержку новых наборов простым пересечениям уже имеющихся наборов. Это происходит за линейное время. Алгоритм пересечения и является определяющей особенностью PPV[4].

1.12. PrePost

год публикации: 2012

подход: «генерация кандидатов и тестирование»

используемые структуры данных:

PPC-tree – префиксное дерево, описанное выше. На рис. 5 приведён пример его построения. Также в переборе задействован N-list, описанный выше.

некоторые ключевые моменты:

Как отмечают сами авторы алгоритмов, PrePost и PrePost+ отличаются только способом отсечения кандидатов: первый использует свойство единственности пути в N-list, в то время как второй основан на свойстве наборов с одинаковой поддержкой, описанном выше[4].

1.13. FIN

год публикации: 2014

подход: «генерация кандидатов и тестирование»

используемые структуры данных:

Node-set – структура аналогичная N-list и Node-list, но использующая вдвое меньший объём памяти, так как она требует хранения только одного параметра pre-order либо post-order вместо обоих сразу.

POC-tree («Pre-Order Coding tree») – префиксное дерево, абсолютно аналогичное PPC-tree, но не хранящее параметр post-order вообще.

Set-enumeration tree – дерево, содержащее все возможные наборы признаков, упорядоченный в порядке не убывания поддержки. Каждая вершина хранит единственный часто встречающийся признак.

некоторые ключевые моменты:

Алгоритм во многом напоминает Pre-Post, отличаясь лишь используемыми структурами данных[4].

2. Методология сравнительного анализа

Следующая методология сравнительного анализа была сформулирована мной с целью сравнения алгоритмов, основанных на эвристиках, что делает бесполезным асимптотическую оценку их сложности и затрачиваемой памяти.

1. Чётко сформулировать условие задачи, которую решают алгоритмы, представленные к сравнению, и не допускать рассмотрения производных задач и задач с искажённым условием.

2. Сформулировать способ поиска и отбора исследований, описывающих алгоритмы сравнительного анализа.

3. Сформировать критерии и способы исследования.

4. Изучить и структурировать все существующие на момент проведения исследования алгоритмы, решающие поставленную задачу, а также работы по сравнению некоторых из них. Выделить общие подходы, если таковые имеются.

5. Сопоставить между собой имеющиеся результаты сравнительных исследований, в том числе проводимых самими авторами алгоритмов при их публикации.

6. Провести собственные эксперименты по сравнению всех имеющихся алгоритмов, учитывая следующие аспекты:

    - проводить сравнение в равных условия: реализация на единой платформе с использованием общего аппаратного и программного обеспечения на одинаковых для всех алгоритмов исходных данных, с единообразным способом доступа к ним;
    - исключить влияние особенностей реализации алгоритмов, способных повлиять на результаты исследования; для этого необходимо, чтобы реализации алгоритмов в точности совпадали с их авторскими описаниями;

- понимать, что любые изменения в исходных алгоритмах порождают новые алгоритмы, требующие отдельной публикации; компетентные исследования формулируются достаточно точно для того, чтобы избежать неоднозначной интерпретации при их реализации;
- обеспечить достаточное многообразие исходных данных для объективного сравнения показателей производительности.

7. Сопоставить результаты собственного всеобъемлющего эксперимента с совокупным результатом предыдущих работ.

8. Изучить зависимость быстродействия алгоритмов от характеристик исходных данных и создать процедуру выбора оптимального алгоритма.

3. Сравнительный анализ

3.1. Формализация условия задачи

Дан упорядоченный набор признаков $I = \{i_1, i_2, ..., i_m\}$, а также набор исходных транзакций (описаний) $T^n = \{\tau_1, \tau_2, ..., \tau_n | \tau \subset I\}$, для каждого набора признаков $s \subset I$ положим $s(\tau) = 1$, если признаки из s совместно встречаются в $\tau \in T^n$.

Поддержкой (support) набора s назовём $v(s) = \frac{1}{n} \sum_{i=1}^{n} s(\tau_i)$.

Набор $s \subset I$ назовём частым (frequent itemset), если его поддержка превышает минимальную поддержку — входной параметр δ (MinSupp), то есть $v(s) \geq \delta$. Задача состоит в нахождении всех частых наборов.

В данной работе рассматриваются только алгоритмы, решающие задачу поиска частых наборов в первозданной виде, в котором она была описана первоначально[1]. Существуют иные задачи, являющиеся производными от данной, одна из которых – задача поиска обобщённых ассоциативных правил. В ней между признаками вводится некоторая иерархия, они обобщаются в группы, и вся эта информация дополняет исходные транзакции[9]. Ещё одна задача, сформулированная на основе поиска частых наборов, учитывает время совершения транзакций и, соответственно, порядок их совершения. Это изменение требований к исходных данных и формирует новую, отличную от первоначальной задачу. Следовательно, алгоритмы, решающие её, такие как GSP[7], к сравнению не привлекаются.

3.2. Процедура поиска публикаций

Поиск исследований и опубликованных алгоритмов осуществлялся путём перебора по ключевым словам всех существующих статей, индексируемых SCOPUS, Web Of Science, ISSN, ESSN. Каждая найденная работа была тщательно изучена, излишки были отброшены. В результате поиска было найдено 251 исследование, из них 142 рассматривали задачу поиска частых наборов в оригинальной формулировке 1993 года,

при этом из них[4]:

- 36 исследований касались разного рода параллельных реализаций алгоритмов;
- в 13 работах описывались новые алгоритмы;
- в 5 работах некоторые из алгоритмов сравнивались между собой;
- 88 статей описывали различные применения результатов к конкретным задачам;

Немаловажным является тот факт, что любые изменения в существующих алгоритмах, будь то иной способ доступа к исходным данным или их хранения, порождают новый алгоритм. Это происходит, к примеру, с алгоритмами Apriori и Apriori Hybrid[4]. Поскольку эти аспекты учтены авторами решений, а написанные мной реализации в точности соответствуют опубликованным алгоритмам, в экспериментах в данной работе я не варьирую подобные параметры.

3.3. Критерии исследования

Переход от часто встречающихся наборов к ассоциативным правилам у всех алгоритмов осуществляется приблизительно одинаково и с одинаковой временной сложностью. Каждый из них стремится оптимизировать главную вычислительную часть — поиск часто встречающихся наборов. И здесь асимптотическая сложность алгоритмов уже не имеет значения для конечной производительности, поскольку все они используют эвристики[3]. Рассмотрим критерии оптимальности в отдельности:

1. Асимптотика.

   В теории алгоритмов этот показатель играет важную роль, однако в данном случае он совершенно лишён смысла. Дело в том, что все

алгоритмы пользуются некоторыми эвристиками, будь то отсечения перебора или попытки сокращения области поиска. Их эффективность напрямую зависит от конкретных данных, их плотности и равномерности распределения. В подобных случаях иногда рассматривают худший случай и выделяют порядок количества операций исходя из него, но такая ситуация возникает только на искусственных специально подобранных примерах, которые не встречаются в реальных задачах.

2. Простота реализации.

   Сам по себе данный критерий является достаточно субъективным. Этот факт находит подтверждения даже в истории IT-индустрии, ссылаясь на тот период, когда размер жалования инженеров вычисляли, исходя из сложности их работы. Были попытки подсчёта количества строк, времени, отводимого на написание кода, но ни одна из них не принесла желаемых результатов. Поскольку всё зависит от конкретного специалиста, от манеры работы, а не только от самой задачи, критерии простоты сильно размыты. Однако автор алгоритма Relim всё же позволяет себе опираться на данное понятие. Я полагаю его интуитивно понятным, но не принимаю во внимание при сравнительном анализе алгоритмов.

3. Объём затрачиваемой памяти.

   Это достаточно показательный критерий. Измеренный на реальных экспериментах он может являться основанием для выбора наиболее эффективного решения. В данном случае асимптотическая оценка также неприменима по описанным выше причинам. Необходимый для решения объём памяти принят мной как один из параметров оценки. Однако он не является ключевым, поскольку далеко не все авторы алгоритмов в своих работах придают ему значение, а разница затрат рассматриваемых алгоритмов оказалась невелика.

4. Время исполнения.

   Поскольку ключевой фактор – это применение решения к реальным задачам, наиболее важным является именно практический результат, а именно время работы алгоритма. Этот параметр и будет играть главную роль в сравнительном анализе.

3.4. Выделение основополагающих подходов

Алгоритмы поиска частых наборов основываются на двух основных подходах: «генерация кандидатов и тестирование» (candidate-generation-and-test) и «наращивание шаблонов» (pattern-growth).

Основоположником первого подхода стал алгоритмы Apriori, поэтому соответствующие решения называются также Apriori-like алгоритмами. Все они генерируют наборы в некотором порядке (например, в порядке возрастания длины), на каждом шаге отсекая некоторые из них по свойству антимонотонности[4]:

$$\forall s \subseteq I, \forall s` \subset sv(s`) \geq v(s)$$

Поддержка набора признаков не превосходит поддержки любого его поднабора. Действительно, всякий раз, когда встречается набор s, встречается и его поднабор $s`$. Следовательно, частота встречаемости $s$ не превосходит частоту встречаемости $s`$.

«Apriori-like» алгоритмы:

1. Apriori
2. Apriori Hybrid
3. PrePost
4. PrePost+
5. Eclat
6. dEclat

7. PPV

8. FIN

Подход «наращивание шаблонов» впервые был предложен в алгоритме FP-Growth. В нём поиска часто встречающихся наборов осуществляется рекурсивно: исходные данные разбиваются на части по некоторому признаку, в каждой из частей ищутся локальные ответы, которые в дальнейшем наращиваются до окончательного результата[4].

«FP-Growth-like» алгоритмы:

1. FP-Growth

2. Relim

3. LCMFreq v.2/v.3

4. H-mine

3.5. Анализ существующих результатов

Поскольку все алгоритмы основаны на эвристиках, их производительность зависит от характеристик исходных данных, а не только от их объёма. Следовательно, единственным способом сравнения подходов становится эксперимент.

Несмотря на то, что единовременного сравнения всех имеющихся на сегодняшний день алгоритмов до сих пор не проводилось, авторы каждого нового решения экспериментально сравнивают свой алгоритм с несколькими уже опубликованными. Помимо этого существует несколько сравнительных исследований, в каждом из которых участвует по несколько алгоритмов. Несмотря на то, что каждый конкретный эксперимент проводился в собственных, отличных от других условия, имеет смысл объединить результаты, так как внутри каждого сравнения однородность условий была соблюдена.

Для создания общей картины, отражающей совокупность предыдущих сравнений, введём бинарное отношение превосходства в производительности между двумя алгоритмами и составим на его основе графы затрат времени (рис. 6) и памяти (рис. 7) соответственно.

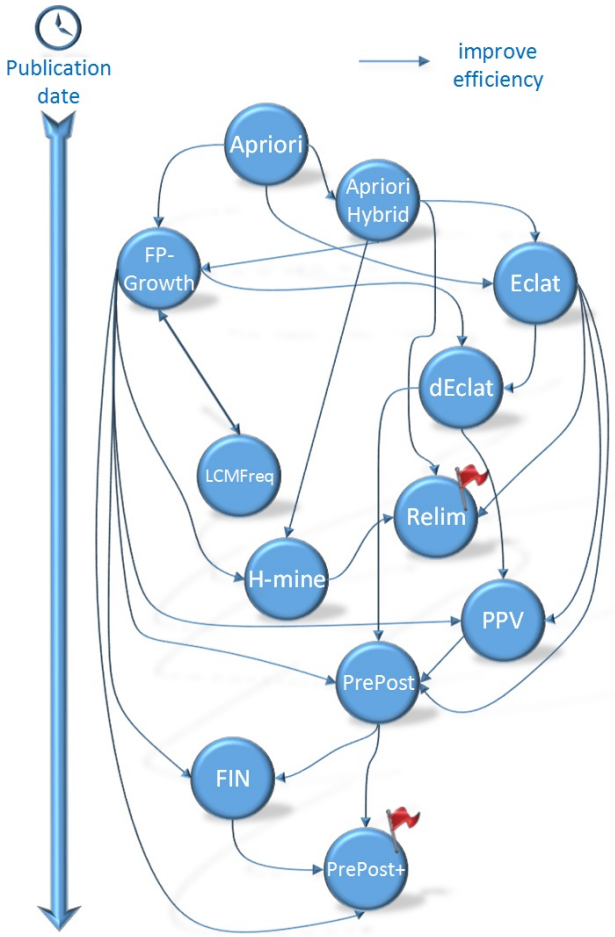

Рис. 6: Граф результатов попарного сравнения производительности алгоритмов. В вершинах графа находятся алгоритмы. Рёбра графа ориентированы по направлению от медленных алгоритмов к быстрым. Красными флажками выделены непревзойдённые в данных сравнениях алгоритмы[4].

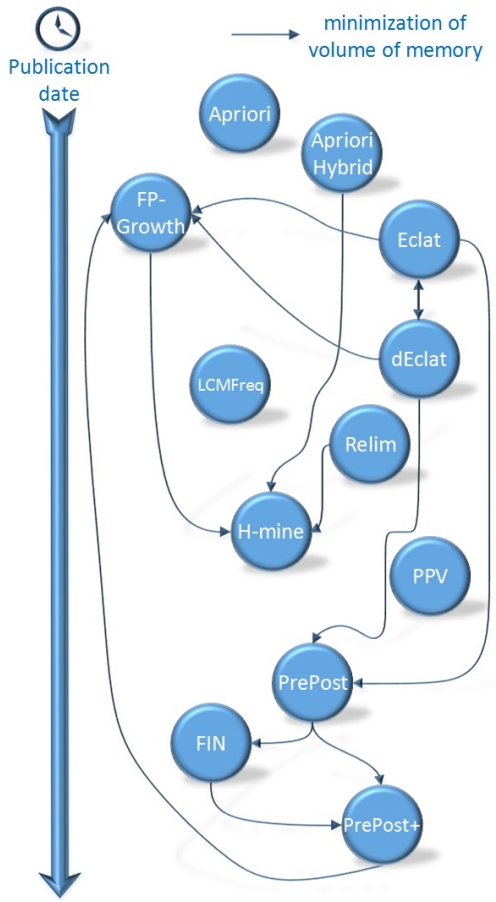

Рис. 7: Граф результатов попарного сравнения объёмов затрачиаемой алгоритмами памяти. Рёбра графа ориентированы по направлению уменьшения затрат памяти[4].

Приведённые выше схемы (рис. 6, 7) описывают всю картину в совокупности, поскольку все эксперименты проводились на открытых базах данных, таких как Pumsb, Mushroom, Connect, Chess, Accidents и тому

подобное[4]. Pumsb содержит результаты переписи, Mushroom – характеристики видов грибов, Connect и Chess основаны на информации о ходе соответствующих игр, Accidents – на информации о дорожном движении. Таким образом, введённое отношение является транзитивным и, как следствие, позволяет осуществить комплексный сравнительный анализ всех алгоритмов. Стоит отметить, что существует вырожденное отношение, отмеченное двойной стрелкой. Оно означает, что в зависимости от предложенных входных данных соответствующие решения демонстрируют приблизительно равные результаты, и их принято считать равноценными[4].

Граф, иллюстрирующий относительные затраты памяти, разряжен. Это объясняется тем, что далеко не все авторы алгоритмов и исследований придают значение объёму используемой памяти.

3.6. Эксперименты

Граф на рис. 6 построен на основе обособленных результатов, следовательно, не гарантирует высокую степень достоверности на стыке разных экспериментов. Для получение наиболее точной картины необходимо было провести единые эксперименты в одинаковых для всех алгоритмов условиях. Полученное мной сравнение выгодно отличается от уже существующих по следующим параметрам:

1. одновременно сравнивались все имеющиеся на сегодняшний день алгоритмы поиска часто встречающихся наборов;

2. все решения реализованы на одном и том же языке программирования – Java EE 8;

3. реализация каждого алгоритма в точности соответствует авторской публикации;

4. все эксперименты проводились на одной аппаратной платформе: процессор Intel(R) Core(TM) i7-4700HQ CPU @ 2.40GHz, ОЗУ 12.0 Гб;

5. доступ к физической памяти был одним и тем же во всех случаях;

6. исходные данные были общими для всех алгоритмов: набор данных Mushrooms, средняя длина транзакции – 23, размер словаря признаков – 119 элементов, общее количество транзакций – 8124; данные содержат описания различных видов грибов;

Значение минимальной поддержки оказывает значительное влияние на количество наборов в ответе, их длину и, следовательно, время работы алгоритма. Варьируя входной параметр поддержки, я получил следующие результаты (рис. 8, 9).

В итоге можно построить следующий рейтинг:

1. Relim
2. PrePost+
3. FIN
4. PrePost
5. PPV
6. H-mine
7. dEclat
8. FP-Growth
9. LCMFreq
10. Eclat
11. Apriori Hybrid
12. Apriori

Алгоритмы Relim и PrePost+ показали сравнительно близкие результаты.

Minimal support	Relim	PrePost+	FIN	PrePost	PPV	H-mine	dEclat	FP-Growth	LCMFreq	Eclat	Apriori Hybrid	Apriori
70%	397,62	401,15	412,45	415,23	417,56	419,35	426,12	427,89	429,96	434,92	437,53	438,24
75%	372,18	373,32	383,78	385,11	389,98	393,02	403,43	404,14	409,07	416,07	419,23	422,91
80%	341,12	351,87	362,91	365,12	373,32	374,79	379,81	377,02	382,59	391,19	392,98	395,01
85%	332,89	337,6	349,45	348,56	358,76	360,16	373,14	375,51	380,31	390,69	394,11	394,09
90%	327,92	328,12	340,11	345,61	349,08	350,38	359,26	362,97	364,82	375,19	378,85	379,02

Рис. 8: Таблица результатов сравнительного эксперимента с базой Mushrooms. Время работы алгоритмов указано в секундах[4].

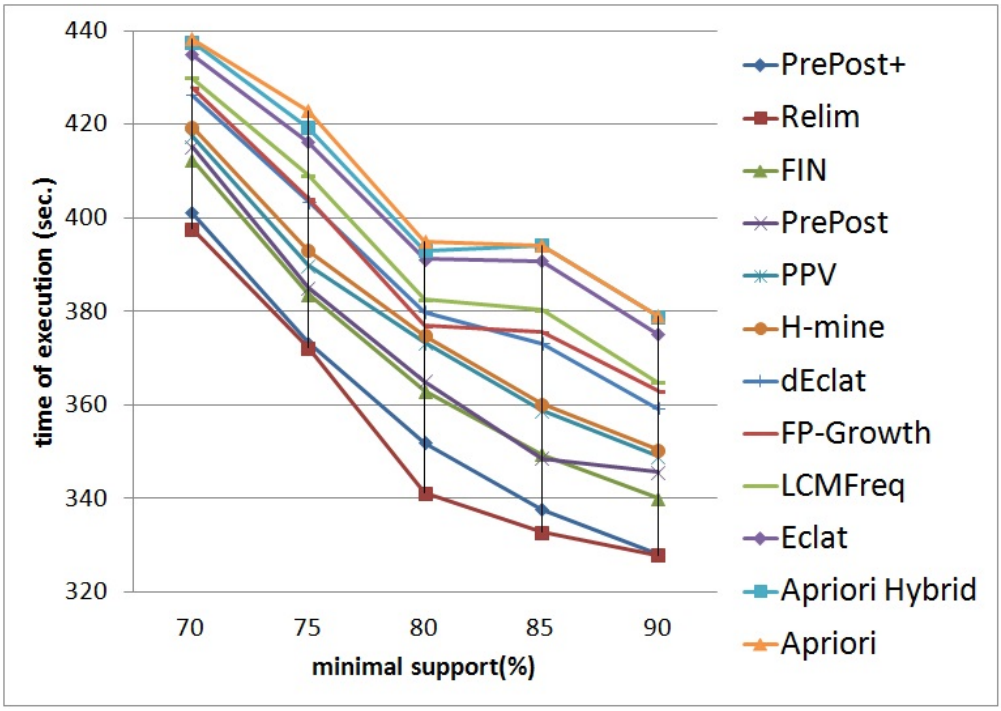

Рис. 9: График зависимостей времени работы алгоритмов с базой Mushrooms от значения минимальной поддержки[4].

Провести эксперименты с меньшими параметрами минимальной поддержки не позволили малые вычислительные мощности. Однако это никак не отразилось на результатах сравнения, так как использованный диапазон значений поддержки достаточно репрезентативен.

3.7. Алгоритм выбора оптимального подхода

Все алгоритмы, решающие задачу поиска часто встречающихся наборов, используют разного рода эвристики, что делает их крайне за-

висимыми от исходных данных. Несмотря на то, что в общем экспериментальном сравнении (рис. 8, 9) алгоритм Relim оказался наиболее эффективным, Relim и PrePost+ показали близкие результаты, являясь при этом представителями разных фундаментальных подходов: «наращивание шаблонов» и «генерация кандидатов и тестирование» соответственно. В связи с этим, возникла необходимость провести более глубокий анализ, перейдя от общего сравнения решений между собой к выявлению зависимостей быстродействия алгоритмов от характеристик исходных данных[3].

Для выявления данной зависимости были проведены аналогичные эксперименты с другой открытой базой данных, после чего сопоставим результаты сравнения алгоритмов и характеристики исходных источников информации. Для следующей серии экспериментов была выбрана общедоступная база Chess, содержащая результаты шахматных партий вместе описаниями ходов игр. Средняя длина транзакции – 6, словарь признаков состоит из 36 элементов, общее количество транзакций – 28056[3]. В итоге были получены следующие результаты (рис. 10, 11).

Minimal support	Relim	PrePost+	FIN	PrePost	PPV	H-mine	dEclat	FP-Growth	LCMFreq	Eclat	Apriori Hybrid	Apriori
70%	807,2	788,2	825,8	838,8	833,6	840,8	853,2	854,2	858,4	880,6	874,42	883,42
75%	740,64	735,8	753,9	778,9	773,23	787,8	804,4	816,82	818,14	832,14	838,46	845,82
80%	703,74	682,24	723,8	742,2	736,64	749,6	759,62	762,04	765,21	782,39	785,66	790,02
85%	678,12	675,43	698,9	709,1	705,52	720,5	746,29	751,02	760,31	780,46	788,98	789,09
90%	648,24	647,88	672,3	672,2	690,16	700,8	710,52	716,1	724,92	740,47	751,85	754,02

Рис. 10: Таблица результатов сравнительного эксперимента с базой Chess. Время работы алгоритмов указано в секундах[3].

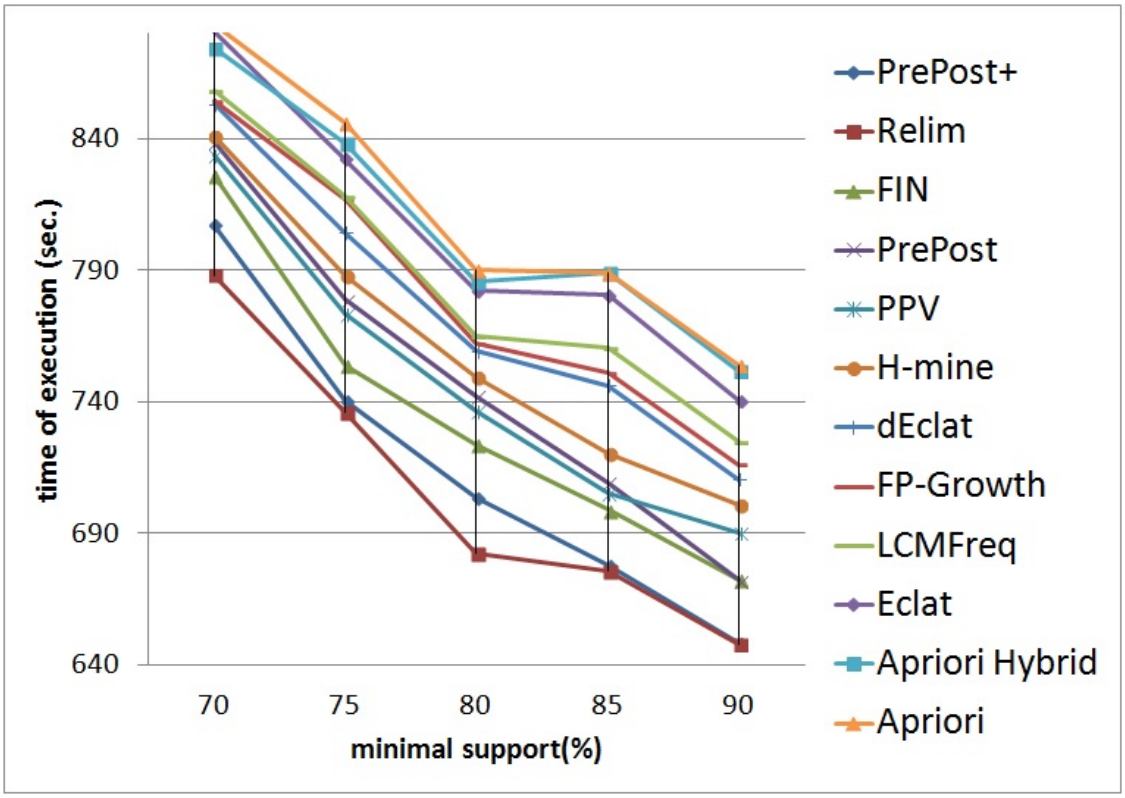

Рис. 11: График зависимостей времени работы алгоритмов с базой Chess от значения минимальной поддержки[3].

В результате эксперимента различий в относительной скорости работы алгоритмов обнаружено не было. Задавшись вопросом поиска общих характеристик наборов данных Mushrooms и Chess, я установил, что у них близки значения показателя, названного мной «покрытием словаря». Под этим термином я подразумеваю количество уникальных признаков транзакции по отношению к мощности словаря:

$$average\ cover\ of\ a\ glossary = \frac{1}{n}\sum_{j=1}^{n}\frac{|\tau_j|}{|D|} \times 100,$$ где n – количество транзакций, D – словарь, τ_j – транзакции, $|*|$ – мощность соответствующего множества; предполагается, что в транзакциях и в словаре признаки содержатся без повторений[3].

У баз данных Mushrooms и Chess средние значения показателей покрытия словаря равны 23.7% и 22.1% соответственно. В экспериментах с использованием этих наборов данных наиболее эффективным алгоритмом признан Relim[3]. Таким образом была сформулирована гипо-

теза: «При показателях среднего значения покрытия словаря порядка 20% FP-Growth-like алгоритмы эффективнее, чем Apriori-like алгоритмы».

Поскольку имеющиеся в открытом доступе данные не располагают достаточным разнообразием для проверки и развития данной гипотезы, а подбор баз данных под конкретные значения среднего покрытия словаря, его мощности и количества транзакций в наборе занимает большое количество времени, мной был создан инструмент, позволяющий генерировать случайный dataset по этим трём заданным входным параметрам.

Для развития гипотезы о зависимости быстродействия алгоритмов от среднего покрытия словаря исходных данных, я выбрал одно для всех экспериментов значение минимальное поддержки, равное 80%, поскольку именно с этим параметром различия во времени работы между алгоритмами Relim и PrePost+ были наиболее значительны. В данном особенно важно сравнение именно этих двух алгоритмов, так как каждый из них является наиболее эффективным представителем своего фундаментального подхода. Прочие параметры были выбраны, исходя из имеющихся вычислительных ресурсов: мощность словаря – 70 признаков, количество транзакций – 90000. Эксперименты были проведены с использованием того же аппаратного и программного обеспечения, что и в параграфе 3.6. В результате были проведены следующие замеры времени работы (рис. 12, 13, 14, 15).

Avg cover of a glossary	PrePost+	Relim	FIN	PrePost	PPV	H-mine	dEclat	FP-Growth	LCMFreq	Eclat	Apriori Hybrid	Apriori
2%	2104,2	2356,5	2197	2188,6	2191,3	2446,3	2211,3	2516,6	2552	2381	2390,4	2461,4
5%	2168,3	2294,4	2232	2215,9	2206,9	2374,6	2226,9	2445,3	2471,9	2387	2414,3	2493,3
10%	2237,5	2263,1	2280	2261,6	2265,6	2306,1	2269,6	2405,9	2423,8	2420	2452,1	2551,1
15%	2255,7	2192,7	2271	2249,4	2255,4	2237,8	2275,4	2310,72	2350,4	2441	2462,7	2574,7
25%	2270,3	2208,5	2299	2312,4	2324,4	2340,7	2345,4	2347,9	2363,7	2479	2491,8	2590,8

Рис. 12: Таблица результатов сравнительного эксперимента с искусственно полученной базой данных. Время работы алгоритмов указано в секундах[3].

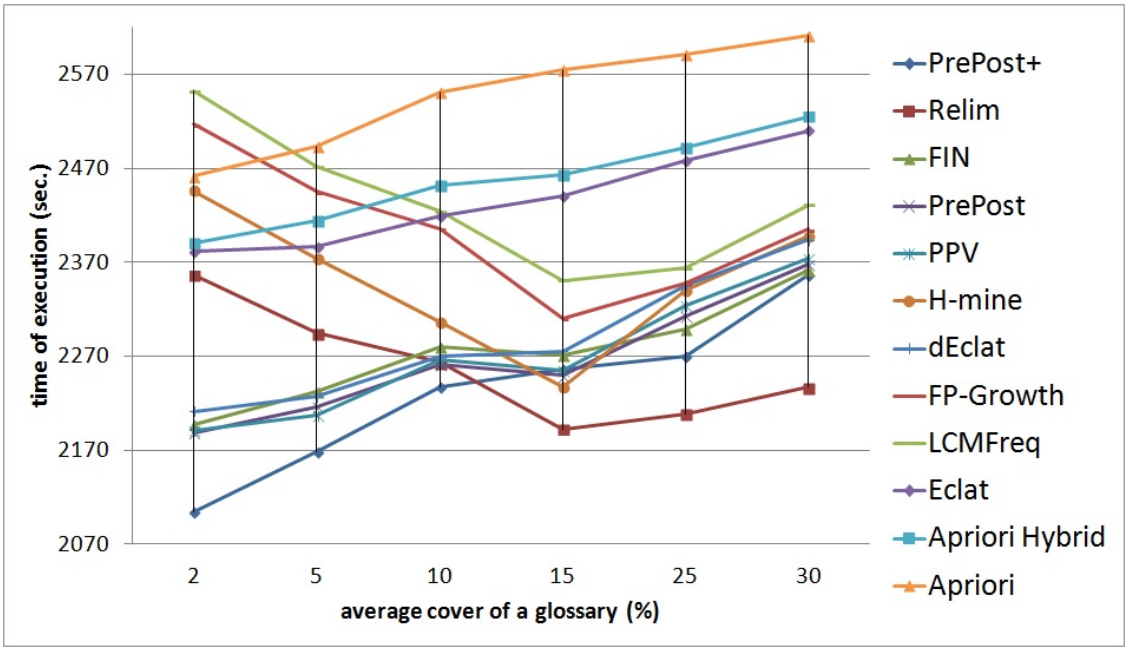

Рис. 13: График зависимостей времени работы алгоритмов с искусственно созданной базой данных от среднего значения покрытия словаря[3].

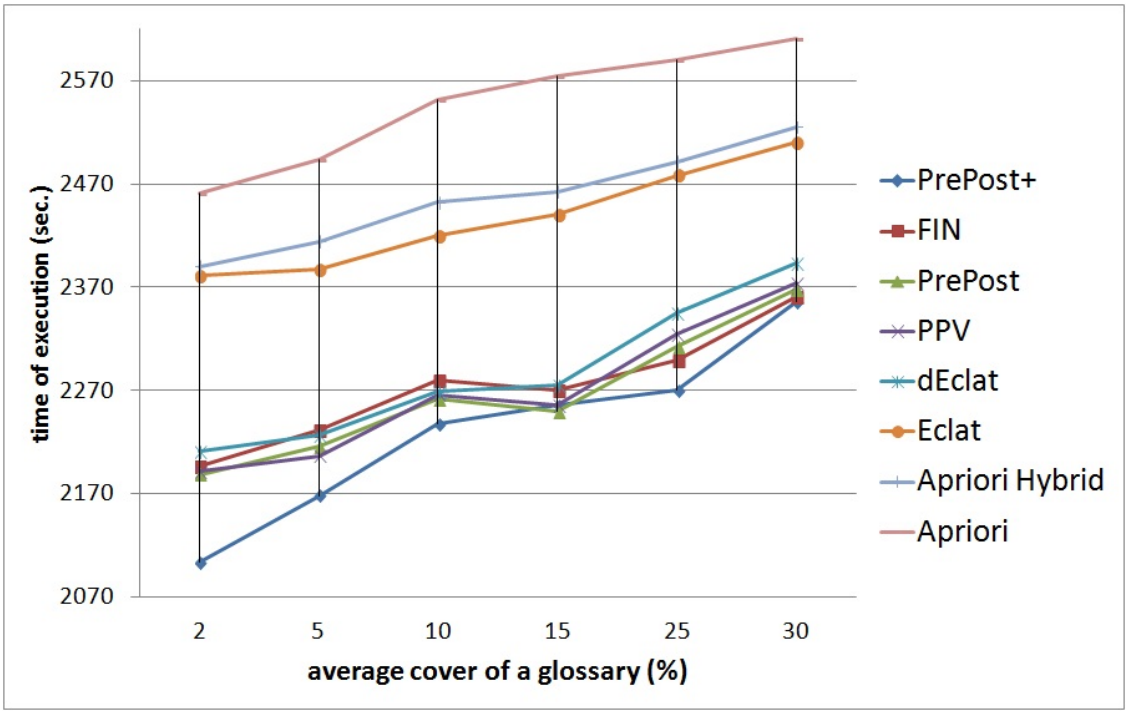

Рис. 14: График зависимостей времени работы Apriori-like алгоритмов с искусственно созданной базой данных от среднего значения покрытия словаря[3].

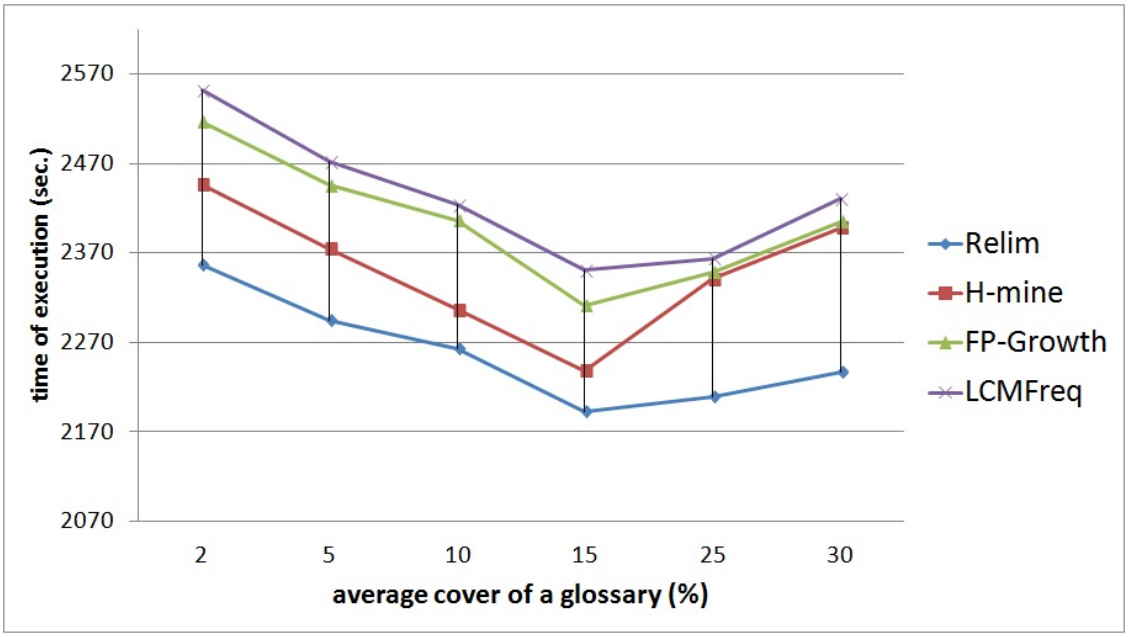

Рис. 15: График зависимостей времени работы FP-Growth-like алгоритмов с искусственно созданной базой данных от среднего значения покрытия словаря[3].

4. Анализ результатов

В ходе сравнительного анализа, согласно созданной мной методологии, были исследованы все имеющиеся на сегодняшний день алгоритмы, решающие задачу поиска частых наборов в оригинальной формулировке [1]. Основным критерием сравнения было выбрано время работы алгоритмов по причине их эвристической природы.

Все существующие решения были распределены по двум группам согласно фундаментальным подходам, лежащим в их основе: «наращивание шаблонов» и «генерация кандидатов и тестирование».

Опубликованные частичные сравнения, проведённые авторами алгоритмов, были проанализированы и объединены в общую картину, выраженную в виде графа (рис. 6). Несмотря на то, что в нём отсутствуют циклы, то есть не содержится противоречий, в основе обобщённого результата лежат обособленные эксперименты, проведённые в различных условиях: программное и аппаратное обеспечение, исходные данные, язык реализации. Более того, многие алгоритмы не попали в прямое сравнение друг с другом. Всё это ставит под сомнение итоговую точность[4].

Для получение качественного результата были проведены эксперименты с соблюдением единых для всех условий. В итоге были выявлены два наиболее эффективных алгоритма с точки зрения временных затрат на исполнение: Relim и PrePost+[4]. Важным оказалось то, что они являются представителями двух разных подходов: «Candidate-generation-and-test» и «pattern-growth».

Следующим шагом исследования стали сравнительные эксперименты с целью поиска взаимосвязи между характеристиками исходных данных. Я обнаружил метрику, «среднее покрытие словаря», которая легла в основу критерия выбора наиболее вычислительно эффективного подхода в зависимости от характеристик исходных данных: при низких показателях этого индикатора, 2%–10%, оптимальным будет подход «Candidate-generation-and-test»; при высоких значениях, 15%–30%, нужно выбрать представителя подхода «pattern-growth»(рис. 14, 15).

Данный факт можно объяснить тем, что при малых значения среднего покрытия словаря эвристики эффективно отсекают кандидатов при переборе, что заметно ускоряет работу алгоритмов. Однако с возрастанием значений контрольной величины отсечение срабатывает всё позже, что пагубно сказывается на эффективности Apriori-like алгоритмов. Высокая эффективность подхода FP-Growth-like в диапазоне величины среднего покрытия словаря от 15% до 30% объясняется тем, что локальных ответов немного, они наращиваются за меньшее количество шагов по причине малых отличий транзакций друг от друга. Побочным результатом в данном случае стало создание инструмента для генерации случайных наборов данных по заданным параметрам: количество транзакций, мощность словаря и величина его среднего покрытия.

Таким образом в данном исследовании я сформировал критерий выбора оптимального подхода в зависимости от характеристик исходных данных и нашёл наиболее эффективный алгоритм среди существующих на сегодняшний день в каждом подходе. Если в будущем будут опубликованы новые алгоритмы в каком то из подходов, то полученное мной правило останется неизменным.

5. Заключение

Итак, в настоящей работе была решена актуальная на сегодняшний день задача проведения сравнительного анализа алгоритмов поиска часто встречающихся наборов с целью выявления наиболее эффективного из них. До появления данного исследования никто не проводил единовременного сравнения всех существующих алгоритмов в равных условиях.

Помимо этого, я сформировал методологию сравнительного анализа, которая может быть применена по отношению к любым эвристическим алгоритмам.

Важным итогом работы стало появление критерия выбора вычислительно оптимального алгоритма, исходя из характеристик исходных данных. Этот результат может быть применён в любой задаче, использующей поиск часто встречающихся наборов. Одна из таких задач, положившая начало данному исследованию, была решена – я построил профили заведений крупной сети ресторанов Coffeemania. Результаты работы были внедрены в реальное предприятие.

Список литературы

[1] Agrawal Rakesh, Imielinski Tomasz, Swami Arun. Mining associations between sets of items in large databases. ACM SIGMOD International Conference on Management of Data. — SIGMOD, 1993. — P. 207–216.

[2] Busarov Vyacheslav, Grafeeva Natalia. The solution of the profiling problem based on Data Analysis. Conference of Open Innovation Association the 19th, FRUCT. — IEEE, 2016. — P. 307–312.

[3] Busarov Vyacheslav, Grafeeva Natalia, Mikhailova Elena. The Choice of Optimal Algorithm for Frequent Itemset Mining. Frontiers in Artificial Intelligence and Applications. — IOS Press, 2016. — P. 211–224.

[4] Busarov Vyacheslav, Grafeeva Natalia, Mikhailova Elena. A Comparative Analysis of Algorithms for Mining Frequent Itemsets. Communications in Computer and Information Science. — SPRINGER, 2016. — P. 136–150.

[5] Cohen Marc-David, Parks Judith. Cross-selling optimizer. — Patent right US20020116237 A1, 2002.

[6] Gangurde Roshan, Kumar Dr. Binod, Gore Dr. S. D. Building Prediction Model using Market Basket Analysis. International Journal of Innovative Research in Computer and Communication Engineering. — IJIRCCE, 2017. — P. 2541–2548.

[7] Huang Jen-Wei, Tseng Chi-Yao, Ou Jian-Chih. A General Model for Sequential Pattern Mining with a Progressive Database. Transactions on Knowledge and Data Engineering 20(9). — IEEE, 2008. — P. 1153–1167.

[8] Ilayaraja M, Meyyappan Thiru. Efficient Data Mining Method to Predict the Risk of Heart Diseases Through Frequent Itemsets. Procedia Computer Science. — ELSEVIER, 2015. — P. 586–592.

[9] Sriphaew Kritsada, Theeramunkong Thanaruk. Mining Generalized Closed Frequent Itemsets of Generalized Association Rules. Lecture Notes in Computer Science. — SPRINGER, 2003. — P. 476–484.

[10] Wong Ke Wang, Zhou Senqiang, Yang Qiang. Mining Customer Value: From Association Rules to Direct Marketing. Data Mining and Knowledge Discovery. — SPRINGER, 2005. — P. 57–79.